НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра автоматизованих систем обробки інформації і управління

«На правах рукопису»

УДК 004.053

«До захисту допущено»
В.о. завідувача кафедри
_____ *О.А.Павлов*
(підпис) (ініціали, прізвище)
“___” _____2019 р.

Магістерська дисертація

зі спеціальності *121 «Інженерія програмного забезпечення»*

на тему: «Алгоритмічне та програмне забезпечення для автоматизованого створення інформаційних систем на основі метаданих»

Виконала: студент VI курсу, групи *ІП-82мп*

Нечай Дмитро Олександрович _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник *к.т.н., доцент Баклан І.В.* _____
(посада, науковий ступінь, , вчене звання, прізвище та ініціали) (підпис)

Консультант *доц., к.т.н., Ліщук К.І.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент *проф.каф.ОТ,д.ф.-м.н.,с.н.с. Гордієнко Ю.Г.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»

Факультет *інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації та управління*
(повна назва)

Рівень вищої освіти *другий (магістерський) за освітньо-професійною програмою*

Спеціальність *121 «Інженерія програмного забезпечення»*
(код і назва)

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *О.А. Павлов*
(підпис) (ініціали, прізвище)

«___» _____ 20*19* р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Нечаю Дмитру Олександровичу
(прізвище, ім'я, по батькові)

1. Тема дисертації *Алгоритмічне та програмне забезпечення для автоматизованого створення інформаційних систем на основі метаданих*

науковий керівник дисертації *к.т.н., доцент Баклан І.В.*.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від "____" _____ 20*19* р. № _____

2. Строк подання студентом дисертації "____" _____ 20*19* р.

3. Об'єкт дослідження *процес розробки розподілених інформаційних систем*

4. Предмет дослідження *автоматизація створення розподілених інформаційних систем на основі метаданих*

5. Перелік завдань, які потрібно розробити _____
Спроектувати мову метаданих, створити інтерпритатор такої мови, що дозволить автоматизувати генерації інформаційних систем, та гарантувати можливість

масштабування таких систем

6. Перелік графічного матеріалу

1)Схема структурна зв'язків між сутностями в програмному забезпеченні.

2) Схема структурна бази даниз

7. Орієнтовний перелік публікацій *Виявлення об'єктів у просторі за допомогою глибинного навчання; Мережеві протоколи для передачі даних у застосунках доповненої реальності*

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання *"_01" вересня* 20*19* р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1	*Формалізація результатів дослідження*	*20.09.2019*	
2	*Аналіз вимог та проектування архітектури*	*10.10.2019*	
3	*Розробка мови метаданих та її інтерпритатора*	*20.10.2019*	
4	*Розробка модуля генерації серверної частитни*	*25.10.2019*	
5	*Розробка модуля генераціх користувацьких інтерфейсів*	*15.11.2019*	
7	*Тестування програмного забезпечення*	*20.11.2019*	
8	*Оформлення документації*	*01.12.2019*	
9	*Подання роботи на попередній захист*	*05.12.2019*	
10	*Подання роботи на основний захист*	*16.12.2019*	

Студент _____ (підпис) _____ (ініціали, прізвище)

Науковий керівник _____ (підпис) _____ (ініціали, прізвище)

РЕФЕРАТ

Актуальність теми: зі збільшенням запиту на автоматизацію процесів -- збільшується запит на створення велокої кількості типових інформаційних систем, а потреба в публікації великої кількості відкритих даних призводить до необхідності в масштабуванні таких систем.

Мета дослідження: дослідити можливість створення мови метаданих для автоматизованої генерації інформаційних систем на її основі.

Для реалізації поставленої мети були сформульовані **наступні завдання**: Спроектувати мову метаданих, створити інтерпритатор такої мови, що дозволить автоматизувати генерації інформаційних систем, та гарантувати можливість масштабування таких систем.

Об'єкт дослідження: процес розробки розподілених інформаційних систем.

Предмет дослідження: автоматизація створення розподілених інформаційних систем на основі метаданих.

Методи дослідження: дослідження, аналіз, експеримент.

Наукова новизна: Найбільш суттєвими науковими результатами магістерської дисертації є: розроблена мова метаданих та механізм горизонтального розподілення систем.

Практичне значення отриманих результатів визначається тим що запропоноване програмне забезречення спрощує та пришвидшує розробку типових розподілених інформаційних систем.

Зв'язок роботи з науковими програмами, планами, темами: Робота виконувалась на кафедрі автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського» в рамках теми «Інтелектуальні методи програмування, моделювання і прогнозування з використанням ймовірністного і лінгвістичних підходів». Державний реєстраційний номер 0117U000926.

Апробація: Основні положення роботи доповідались і обговорювались на 3 всеукраїнській науково-практичній конференції молодих вчених та студентів «Інформаційні системи та технології управління».

Публікації: Наукові положення дисертації опубліковані в матеріалах 3 всеукраїнської науково-практичної конференції молодих вчених та студентів «Інформаційні системи та технології управління» та всеукраїнської науково-практичної конференції молодих вчених та студентів «Інформатика та обчислювальна техніка – ІОТ 2016».

Ключові слова: МЕТАДАНІ, ПОМ, РОЗПОДІЛЕНІ, ГЕНЕРАЦІЯ.

ABSTRACT

Topicality: with the increasing demand for process automation – the demand for creating a large number of typival information systems is increasing, and the the need to publish a large amount of open data leads to the need to scale such systems.

Purpose of the research: to develop software for automated creation of information systems baed on metadata.

To achieve this goal, **the following tasks** were formulated: research existing metadata languages, determine their strength and weaknesses, research the software that is currently being used to solve the tasks; propose metadata language, on the basis of which the information system will be generated; develop software that will create client interfaces, inter-program interaction layer, data access layer and persistent storage based on the developed metadata language; guarantee horizontal and vertical scaling of the results IPs.

Object of research: the process of developing distributed information system.

Subject of research: creation of distributed information systems based on metadata.

Research methods: research, analysis, experiment.

Scientific Novelty: the mot significant results of the master's thesis are – proposed metadata language and mechanism of horizontal sitribution of systems.

The practical value of the results is determined by the fact that the proposed software simplifies and accelerates the development of typical distributed information systems.

Relationship with working with scientific programs, plans, topics: the work was perfomed at the department of Automated Information Processing and Management Systems of the National Technical University of Ukraine "Igor Sikorsky Kyiv Politechnic Institute" within the topic "Intelligent Programing, Modeling and Forecasting Methods Using Probabilistic and Linguistic Approaches". State Registration Number 011U000926.

Testing: The main provisions of the work were reported and discussed at the 3rd All-Ukrainian Scientific and Practical Conference of Young Scientists and Students "Information Systems and Management Technologies".

Publications: The scientific provisions of the dissertation are published in the materials 3 all-Ukrainian scientific-practical conference of young scientists and students "Information systems and management technologies" and all-Ukrainian scientific-practical conference of young scientists and students “Informatics and computers – ICT2016”.

Keywords: METADATA, DSL, DISTRIBUTED, GENERATION.

Пояснювальна записка
до магістерської дисертації

на тему: «Алгоритмічне та програмне забезпечення для автоматизованого створення інформаційних систем на основі метаданих»

Київ – 2019 року

ЗМІСТ

ВСТУП

В сучасному світі все більше зростає необхідність використання інформаційних систем на підприємствах незалежно від їх розмір. Кожне підприємство потребує способів автоматизації процесів, контролі та управління ремурсами та комунікації зі споживачем. Розробка інформаційної системи на замовлення є дуже складним процесом і занадто дорогим для малого та середнього бізнесу. Використання існуючих типових рішень також не завжди себе виправдовує: занадто велика ціна на інсталяцію, неможливість масштабування таких рішень, недостатня їх гнучкість або неможливість власноручної їх налаштування, модифікування та підтримки, — все це призводить до великих затрат з боку підприємств.

Із збільшенням суспільного запиту на публікацію відкритих даних у вільний доступ держави створюють все більше сервіс для надання доступ до реєстрів даних, проте створення і підтримка кожної такої системи обходиться державі у велику суму.

В зв’язку з наведеними вище причинами було вирішено створити ПЗ для вирішення всіх описаних проблем.

Мета роботи — розробити мову опису метаданих та створити ПЗ для автоматизації побудови ІС на основі розробленної мови.

Для досягнення мети було поставлено наступні задачі:

дослідження існуючих мов метаданих, виявлення їх переваг та недоліків;

1) дослідження програмних комплексів, які наразі використовуються для вирішення поставлених задач;
2) створення мови метаданих, на основі якої відбуватиметься генерація інформаційної системи;
3) розробка програмного забезпечення, яке буде створювати клієнтські інтерфейси, шар міжпрограмної взаємодії, шар доступу до даних та їх збереження на основі розробленої мови метаданих;

4) гарантувати можливість горизонтально та вертикального масштабування результуючих ІС.

Об’єкт дослідження – процес розробки розподілених інформаційних систем.

Предмет дослідження – автоматизація створення розподілених інформаційних систем на основі метаданих.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У даному розділі розглянуто теоретичні відомості про існуючі методи створення інформаційних систем, існуючі формати опису метаданих та існуючі прогграмні рішення.

1.1 Загальні положення

Інформаційна система (ІС) - система, призначена для зберігання, пошуку та обробки інформації, і відповідні організаційні ресурси (людські, технічні, фінансові та т. Д.), які забезпечують і поширюють інформацію [1].

Корпоративна ІС — автоматизує всі бізнес-процеси цілого підприємства (організації) або їх значну частину, досягаючи їх повної інформаційної узгодженості, беззбиточності і прозорості [2].

Структура проблемно-орієнтованої інформаційної системи може бути описана десяткою [3]:

<S, I, S-I, I-I, P, I-P, P-I, O, P-O, O-O>

де S — множина ситуацій, які можуть ідентифікуватися інформаційною системою; I — множина інформаційних об’єктів (показники, чисельні характеристики, властивості й т.п.) ; S-I — відображення з множини ситуацій на множину інформаційних об’єктів, однозначно забезпечуючих конкретну ситуацію; I-I — множина взаємозв’язків на множині інформаційних об’єктів; P — множина процедур; I-P — опис вхідних параметрів процедур; P-I — опис вихідних параметрів процедур; O — множина операційних елементів (програмні модулі, системи та т.ін.; P-O — відображення з множини процедур до множини операційних елементів, однозначно забезпечуючих конкретну процедуру; O-O — множина взаємозв’язків на множині операційних елементів.

1.2 Типи доповненої реальності

Наразі індустрія програмного забезпечення налічує велику кількість форматів опису даних, серед найбільш популярних варто виділити такі:

DDL(SQL)

Мова опису даних, яка використовується в реляційних базах даних, приклад синтаксису наведений на рисунку 1.1.

```
CREATE TABLE employees (
    id              INTEGER         PRIMARY KEY,
    first_name      VARCHAR(50)     not null,
    last_name       VARCHAR(75)     not null,
    fname           VARCHAR(50)     not null,
    dateofbirth     DATE            not null
);
```

Рисунок 1.1 — приклад синтаксису SQL

XML та XSD

Розширювана мова розмітки (XML) - це простий, дуже гнучкий текстовий формат, похідний від SGML (ISO 8879). Спочатку створений для вирішення завдань широкомасштабного електронного видавництва, XML також відіграє все більш важливу роль в обміні найрізноманітнішими даними в Інтернеті та інших місцях [4].

XSD ((XML Schema Definition — визначення XML-схеми) — рекомендація Всесвітнього консорціуму веб (W3C), визначає, як формально описати елементи сутності за допомогою документу на підмножині розширюваної мови розмітки XML. Такі схеми використовуються в програмуванні для використання [5].

```xml
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Рисунок 1.2 — приклад синтаксису XSD

JSON Schema

JSON Schema це специфікація для формату на основі JSON для визначення структури даних JSON. Він був написаний в рамках проекту IETF, термін дії якого закінчився в 2011 році. Схема JSON була створення для вирішення таких задач:

опису існуючих форматів даних;

чітка документація, яка була б одночасно простою, для сприйняття людиною та аналізу програмним забезпеченням;

повний опис структури документів.

Приклад синтаксису JSON Schema наведено на рисунку 1.3.

```
{
  "$id": "https://example.com/person.schema.json",
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Person",
  "type": "object",
  "properties": {
    "firstName": {
      "type": "string",
      "description": "The person's first name."
    },
    "lastName": {
      "type": "string",
      "description": "The person's last name."
    },
    "age": {
      "description": "Age in years which must be equal to or greater than zero.",
      "type": "integer",
      "minimum": 0
    }
  }
}
```

Рисунок 1.3 — приклад синтаксису JSON Schema

1.3 Існуючі способи створення інформаційних систем

Існує два розповсюджених способів створення нових інформаційних систем, розробка на замовлення та використання існуючих програмних комплексів для реалізації бажаної логіки.

Розробка інформаційних замовлень відбувається таким чином — підприємство формує або винаймає групу розробників та бізнес аналітиків, які аналізують потреби підприємства та приступають до реалізації всього необхідного функціоналу. Головна перевага такого підходу — результуюча інформаційна система буде розроблена у повній відповідності до вимог та специфіки підприємства, та повний над нею контроль, головним недоліком є дороговизна такого підходу, так як необхідно постійно підтримувати досить чисельний штат персоналу, головною задачею якого буде супровід розробленого програмного забезпечення.

Альтернативний підхід полягає у використанні існуючих програмних комплексів, для створення нової системи. У такому випадку винаймається група аналітиків, які мають досвід роботи з вибраним програмним комплексом.

Вони аналізують необхідності підприємства, і уже відповідно до них підготовлють дані для завантаження в комплекс, або запит до програмістів комплексу, які реалізують дану задачу. До переваг даного рішення відноситься висока швидкість отримання початкового результату, та невеликі витрати на першому етапі. До недоліків — низька гнучкість систем коли необхідно додати функціонал, що відрізняються від типового, передбаченого системою, так так звана ситуація “vendor lock-in”, коли вибраний провайдер послуг є єдиним можливим, перехід до нового провайдер дуже затратним, а послуги з супроводу та модифікації програмного продукту можуть виконувати лише спеціалісти сервісу, сукупність цих факторів перетворює провайдера на локального монополіста, що диктує свої умови надання послуг.

1.4 Альтернативні програмні рішення

Серед проаналізованих рішень, було обрано такі, найбільш популярні:

Парус

Парус-Підприємство 7 - проста і зручна, але в той же час потужна повнофункціональна система, призначена для використання на підприємствах різної галузевої спрямованості. Система поставляється повністю готової до експлуатації. Всі документи і звіти користувач може завантажувати як в Microsoft Office, так і в безкоштовний OpenOffice. Система має модульну побудову. Кожен з модулів об’єднаний в єдину інформаційну базу, що забезпечує прозорість доступу до даних [6].

Рисунок 1.3 — Парус Підприємство

1.5 Розробка вимог до програмного забезпечення

Для визначення вимог до програмного забезпечення необхідно визначити ролі користувачів та їх можливості в системі. Для досягнення мети розробки система повинна містити наступні типи акторів:

— Бізнес-аналітик;

— Адміністратор;

— Користувач.

— Система.

Бізнес-аналітик має доступ до створення, перегляду та оновлення метаданих.

Адміністратор має можливість створювати користувачів, управляти їх правами, проводити аудит та моніторинг активності.

Користувачі мають можливість виконання дій в системі на основі їх прав доступу.

Детальний опис варіантів використання наведено в таблиці 1.1.

Таблиця 1.1 – Опис варіантів використання

ID	Актори	Модуль–назва	Передумови	Опис
UC01	Бізнес-аналітик, Система	Робота зі схемами – Перегляд схем	Систему було розгорнуто	Бізнес-аналітик повинен мати доступ до перегляду існуючих схем даних
UC02	Бізнес-аналітик, Система	Робота зі схемами – Створення схем	Систему було розгорнуто	Бізнес-аналітик повинен мати можливість створення нових схем даних
UC03	Бізнес-аналітик, Система	Робота зі схемами – Оновлення схем	В системі існують схеми даних	Бізнес-аналітик повинен мати можливість модифікації існуючих схем даних
UC04	Бізнес-аналітик, Система	Робота зі схемами – Валідація схем	В системі існують схеми даних	Бізнес-аналітик повинен мати можливість перевірити коректність розроблених ним схем даних
UC05	Бізнес-аналітик, Система	Робота зі схемами – Розширення функціоналу схем	Систему було розгорнуто	Бізнес-аналітик повинен мати можливість додавання нових функціонлаьних сутностей в схеми даних

Продовження таблиці 1.1 – Опис варіантів використання

UC06	Система	Робота зі схемами – Завантаження схем	Систему було розгорнуто	Система повинна бути в стані завантажити і опрацювати коректні схеми даних
UC07	Система	Робота зі схемами – Оновлення схем	В систему завантажили схеми даних	Система повинна бути в стані оновити схеми даних
UC08	Система	Робота з даними – створення табличного простору	В систему завантажили схеми даних	Система повинна гарантувати генерацію коректного табличного простору для постійного сховища даних
UC09	Система	Робота з даними – Оновлення табличного простору	В систему завантажили схеми даних	Система повинна гарантувати коректне оновлення табличного простору для постійного сховища даних
UC10	Система	Робота з даними – Генерація запитів	В систему завантажили схеми даних	Система повинна гарантувати коректну побудову запитів на основі метаданих

Продовження таблиці 1.1 – Опис варіантів використання

UC11	Система	Робота з даними – Шардування даних	В систему завантажили схеми даних	Система повинна гарантувати коректне шардування даних
UC12	Система	Інтерфейс користувача – Генерація інтефрейсу	В систему завантажили схеми даних	Система повинна гарантувати коректність автоматичної побудову інтерфейсу користувача на основі схем даних
UC13	Система	Інтерфейс користувача – Оновлення інтефрейсу	В системі було онорвлено схеми даних	Система повинна гарантувати коректність автоматичного оновлення інтерфейсу користувача
UC14	Користувач, Система	Інтерфейс користувача – Виконання операції	В систему завантажили схеми даних	Користувач повинен мати можливість виконання операці’ відповідно до своїх прав доступу

Продовження таблиці 1.1 – Опис варіантів використання

UC15	Адміністратор, Система	Управління користувачами – Створення користувача	Систему було розгорнуто	Адміністратор повинен мати можливість створення користувачів
UC16	Адміністратор, Система	Управління користувачами – блокування користувача	В системі існують користувачі	Адміністратор повинен мати можливість блокування користувачів
UC17	Адміністратор, Система	Управління користувачами – Створення прав користувача	В системі існують користувачі	Адміністратор повинен мати можливість створювати права користувачів
UC18	Адміністратор, Система	Управління користувачами – Видалення прав користувача	В системі існують користувачі	Адміністратор повинен мати можливість видаляти права користувачів
UC19	Користувач, Система	Управління користувачами – Оновлення паролю	Користувач авторизований	Користувач повинен мати можливість оновлення паролю

Продовження таблиці 1.1 – Опис варіантів використання

UC20	Адміністратор, Система	Аудит – Проведення аудиту	Систему було розгорнуто	Адміністратор повинен мати можливість проводити аудит доступу до даних
UC21	Адміністратор, Система	Аудит – Проведення моніторингу	Систему було розгорнуто	Адміністратор повинен мати можливість проводити моніторинг доступу до даних
UC22	Система	Авторизація – Перевірка прав користувача	Користувач відправив запит на виконання операції	Система повинна перевіряти актуальні права користувача в момент виконання операції
UC23	Користувач, Система	Авторизація – Аутентифікація користувача	В системі існують користувачі	Система повинна надавати можливість аутентифікації користувачів

1.5.1 Розробка функціональних вимог

Опираючись на описані сценарії використання, було розроблено список функціональних вимог, детальний їх опис наведено в таблиці 1.2.

Таблиця 1.2 – Опис функціональних вимог

ID	Модуль	Функціональна вимога	Пріоритет
RQ01	Робота зі схемами	Модуль дозволяє отримати доступ на перегляд схем даних	Високий
RQ02	Робота зі схемами	Модуль дозволяє створювати схеми даних	Високий
RQ03	Робота зі схемами	Модуль дозволяє редагувати існуючі схеми даних	Високий
RQ04	Робота зі схемами	Модуль дозволяє проводити валідацію схем	Високий
RQ05	Робота зі схемами	Модуль повинен детально повідомляти про існуючі проблеми	Високий
RQ06	Робота зі схемами	Модуль повинен дозволяти розширювати функціональність схем	Середній
RQ07	Робота зі схемами	Модуль повинен коректно завантажувати схеми	Високий
RQ08	Робота зі схемами	Модуль не повинен завантажувати не коректні схеми	Високий
RQ09	Робота зі схемами	Модуль повинен коректно оновлювати схеми, після їх редагування	Високий

Продовження таблиці 1.2 – Опис функціональних вимог

RQ10	Робота з даними	Модуль повинен коректно створювати табличний простір для постійного сховища даних	Високий
RQ11	Робота з даними	Модуль повинен коректно оновлювати табличний простір для постійного сховища даних	Високий
RQ12	Робота з даними	Модуль повинен генерувати корректні запити	Високий
RQ13	Робота з даними	Модуль повинен детально повідомляти у разі отримання некоректних запитів	Високий
RQ14	Інтерфейс користувача	Модуль повинен автоматично генерувати інтерфейс користувача	Високий
RQ15	Інтерфейс користувача	Модуль повинен автоматично оновлювати інтерфейс користувача	Високий
RQ16	Інтерфейс користувача	Модуль повинен коректно відображати список доступних операцій	Високий
RQ17	Інтерфейс користувача	Модуль повинен дозволяти виконувати доступні операцій	Високий
RQ18	Інтерфейс користувача	Модуль повинен детально повідомляти про помилики, що виникли в процесі винокання операцій	Високий
RQ19	Управління користувачами	Модуль повинен автоматично створювати суперадміністратора	Високий

Продовження таблиці 1.2 – Опис функціональних вимог

RQ20	Управління користувачами	Модуль повинен дозволяти створювати нових користувачів	Високий
RQ21	Управління користувачами	Модуль повинен дозволяти блокувати існуючих користувачів	Високий
RQ22	Управління користувачами	Модуль повинен дозволяти надавати прива користувачам	Високий
RQ23	Управління користувачами	Модуль неповинен дозволяти надавати прива користувачам на дані, до яких у адміністратора немає доступу	Високий
RQ24	Управління користувачами	Модуль повинен дозволяти видаляти прива користувачам	Високий
RQ25	Управління користувачами	Модуль не повинен дозволяти видаляти прива користувачам на дані, до яких у адміністратора немає доступу	Високий
RQ26	Управління користувачами	Модуль повинен надавати можливість змінювати пароль	Середній
RQ27	Управління користувачами	Модуль не повинен дозволяти встановлювати пароль, вразливий до атак підбору	Середній
RQ28	Аудит	Модуль повинен дозволяти проводити аудит виконаних операцій	Середній
RQ29	Аудит	Модуль не повинен дозволяти отримати доступ до чутливих даних під час проведення аудиту	Середній

Продовження таблиці 1.2 – Опис функціональних вимог

RQ30	Аудит	Модуль повинен дозволяти проводити моніторинг виконаних операцій	Середній
RQ31	Авторизація	Модуль повинен коректно перевіряти актуальні права користувача, перед виконанням кожної операції	Вискоий
RQ32	Авторизація	Модуль не повинен авторизувати операції, на які у користувача немає прав	Вискоий
RQ33	Авторизація	Модуль повиен коректно автентифікувати користувача	Вискоий

1.5.2 Розробка нефункціональних вимог

Програмне забезпечення повинне відповідати наступним нефункціональним вимогам:

— локалізація інтерфейсу – українська для української локалізації пристрою;
— адаптивність інтерфейсу – відображення інтерфейсу користувача повинно адаптуватись до розміру та форми екрана пристрою на якому якого запускають;
— інтуїтивно зрозумілий інтерфейс;
— можливість роботи клієнта без доступу до серверу;
— підтримувана версія платформ: Google Chrome v78, Firefox v70, Safari 13. Edge v44 та вище;
— для передачі даних повинні використовуватись захищені канали передачі даних з використанням TLS;
— система повинна дозволяти лише паролі стійкі до підбору;
— розроблене програмне повинно бути стійким до великих навантажень;

— розроблене програмне забезпечення повинно надавати можливість відновлення системи в разі виникнення неполадок.

Висновки до розділу

В ході аналізу предметної області було виявлено такі головні недоліки існуючих альтернативних програмних рішень: відсутність автоматичної генерації інтерфейсу користувача на основі метаданих, непридатність рішень до масштабування, та низька швидкість модифікації, серед розповсюджених форматів опису метаданих головним недоліком була відсутність можливості ефективно описувати бізнес процеси або розширяти функціонал схем.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі було описано розроблений формат метаданих, способи автоматизованої побудови користувацьких інтерфейсів, шару взаємодії з даними, та спосіб гарантування придатності системи до масштабування.

Для опису структури інформаційної системи може бути використаний апарат алгебри каскадів. Модель предметної області повинна забезпечувати формалізоване представлення (логічне, алгебраічне і т.п.) досліджуємих елементів та їх взаємозв’язків. Припускаємо, що об'єкти предметної утворюють множину $X=\{X_j\}$, на якому задано безліч взаємозв'язків $R=\{R_i\}$, де з кожним взаємозв’язком асоціюється поняття арності – $arn(R_i)$, тобто кількість об'єктів, які приймають участь у факті, описуваному за допомогою цього взаємозв'язку. З точки зору математичної лінгвістики, взаємозв'язок в нашому розумінні, буде являти собою дієслівну снову, яка особливо яскраво виражено в ЕНП-мовах.

На основі множини X і додатково введеної множини $\Theta=\{Q_i\}$, елементи якої назвемо зв’язками, будується множина Y (назвемо її носієм множини каскадів) за наступними правилами:

1) якщо $y_i \in X$, то $y_i \in Y$;
2) якщо $y_1, y_2 \in Y$, $Q_j \in \Theta$ то $y_1 Q_{jy2} \in Y$;
3) $\Lambda \in Y$ (Λ – пусте ім’я).

Описані вище правила призводять до алгебри слів, або термів, однак не в загальній постановці, а в спеціальній формі, зручній для побудови апарату каскадів.

Каскадом називатимемо елемент множини:

$$K = \bigcup_{m=0}^{n01} Y^m \times B(R \times B(Y^{n-m}),$$

Де Y^k – k-кратний декартове множення, B(A) – булеан множини A.

Легко помітити, що K складається із множин:

$$K^m = Y^m \times B(R \times B(Y^{n-m}),$$

Які будемо називати m-основними системами каскадів. Число утворюючих m-основних систем залежить від максимальної арності взаємозв'язків, що входять в множину R.

Будемо використовувати наступне відображення каскаду:

$$K_1 = \{\bar{y}_j^m, \{\{R_1^i, A_{ij1}\}, \dots, \{R_{s_i}^i, A_{ijs_1}\}\}\},$$

де $$A_{ijp} = \{\bar{y}_{ijpk}^{n-m}\}, k = 1, \dots, t_{ijp}$$

$$\bar{y}_{ijpk}^{n-m} = \{y_{m+1}^{ijpk}, \dots, y_n^{ijpk}\},$$

При цьому $R_j \in R(K_1)$, $R(K_1)$ – множина взаємозв'язків каскаду K_1.

Визначено морфізм з K^m в K^{m-1}, властивості якого роблять достатнім розгляд властивостей певнлї m-основної системи, які за розглянутим морфізмом розповсюджуються на інші системи сножини каскадів. Для визначеняя операцій алгебри каскадів використаємо множину зв'язок (.,!-).

При цьому вводяться наступні еквівалентні перетворення складних імен:

1) $y_i\ y_j = y_j\ y_i$;
2) $y_i\ (y_j\ y_k) = (y_i\ y_j)\ y_k$;
3) $y_i \wedge_= y_1$;
4) $y_i\ !\ y_{j=}\ y_j\ !\ y_i$;
5) $y_i\ !\ (y_j\ !\ y_k) = (y_i\ !\ y_j)\ !\ y_k$;
6) $y_i\ !\ \wedge = \wedge$;
7) $y_i\ -\ \wedge = y_i$;
8) $y_i\ -\ (y_j\ !\ y_k) = (y_i\ -\ y_j)\ (y_i - y_k)$;
9) $y_i\ -\ (y_j\ y_k) = (y_i\ -\ y_j)\ !\ (y_i - y_k)$;
10) $y_i\ y_i = y_i$;
11) $y_i\ !\ y_i = y_i$;
12) $y_i\ -\ y_i = y_k$;

Сенс веденох множини зв'язків пояснений нижче.

Об'єднанням каскадів:

$$K_1 = \{\bar{y}_{ij}^m, \{\{R_1^i, A_{ij1}\}, \dots, \{R_{s_i}^i, A_{ijs_1}\}\}\} \text{ та } K_l = \{\bar{y}_p^m, \{\{R_1^i, A_{lp1}\}, \dots, \{R_{s_l}^l, A_{lps_1}\}\}\}$$

буде каскад $K_i \cup K_l = \{\bar{y}_j^m \bar{y}_p^m, \{\{R_t, A_t^{\cup}\}, t \in J_r(k_i) \cup J_r(k_l)\}\}$, де

$$A_t^v = \begin{cases} A_{ijr} & \text{, якщо } R_t \notin R(K_l), R_t = R_t^i; \\ A_{ljr} & \text{, якщо } R_t \notin R(K_i), R_t = R_t^l; \\ A_{ijr} \cup A_{ljr} & \text{, якщо } R_t = R_v^i, R_t = R_v^l. \end{cases}$$

$J_x(K_z)$ – множина індексів сножини взаємозв'язків, на основі яких побудований каскад K_z.

Перетин описаних вище каскадів K_i та K_l буде являти собою каскад $K_i \cap K_l = \{\bar{y}_j^m ! \, \bar{y}_p^m, \{\{R_t, A_t^{\cap}\}, t \in J_r(k_i) \cup J_r(k_l)\}\}$, де

$$A_t^v = \begin{cases} A_{ijr} \cap A_{ljr} \text{, якщо } R_t = R_v^i, R_t = R_v^l. \\ \emptyset \end{cases}$$

Різність каскадів вводиться наступним чином:

$K_i \ominus K_l = \{\bar{y}_j^m - \bar{y}_p^m, \{\{R_t, A_t^{\ominus}\}, t \in J_r(k_l)\}\}$, де

$$A_t = \begin{cases} A_{ijr} \text{, якщо } R_t \in R(K_1), R_t = R_v^i. \\ A_{ijr} \backslash A_{ljr} \end{cases}$$

Універсальним каскадом назвемо $K_v^m = \bigcup_{k_i \in K^m} K_i$.

Пустий каскад має наступну структуру:

$$K^m \Lambda = \{\Lambda, \dots, \Lambda, \{\{R_1, \emptyset\}, \dots, \{R_s, \emptyset\}\}\}, s = card(R).$$

Доповнення каскаду K_1 в системі K^m являє собою каскад, який визначається співвідношенням $\overline{K_i} = K_m^v \ominus K_i$.

В результаті множини каскадів з операціями об'єднання, перетину і доповнення і каскадами одиничним (утворюється як об'єднання всіх одиничних каскадів) і універсальним формують булеву алгебру. Описані вище операції відображають класичний підхід до створення алгебраїчних систем. Введення ж операцій, які розглядаються далі, викликано саме характерними особливостями предметної області проектування АСУ гнучкими продуктивними системами [7, 8].

При використанні апарату числення предикатів першого порядку для реалізації пошуку, наприклад, всіх структурних елементів, які забезпечують дані, нам довелося б багаторазово будувати дедуктивний висновок, вирішуючи паралельно предикатні рівняння. В рамках теорії каскадів це можна здійснити введенням спеціальних операцій, і в першу чергу операцій приєднання і транзитивного замикання.

Як було розглянуто в попередньому розділі, правило продукції по своїй суті - просто програма з одного оператора виду

ЯКЩО <УМОВА> ТО <ДІЯ>

Всі правила, які реалізуються в рамках даної теорії, можуть бути розбиті на дві підгрупи. Одну утворюють правила, в <УМОВА> і <ДІЯ> яких входять лише явні посилання на. існуючу реалізацію каскадної бази знань. Їх назвемо явними, або нуль-правилами. Інша груша представлена так званими параметрическими правилами, тобто <УМОВА> та/або <ДІЯ> містять неявні величини (з точки зору лямбда-числення - вільні змінні). Явні правила є однією з головних характеристик конкретної предметної області, водночас як частина параметричних правил кожет бути спільною для багатьох предметних областей.

Основу <УМОВИ>, що включається в правило продукції, становить предикат існування каскаду :

$$E(K_j) = \begin{cases} 1, \text{якщо } K_j \text{ присутнє в базі знань;} \\ 0. \end{cases}$$

<ДІЇ> утворюються у вигляді функціональних виразів над каскадної базою знань, при цьому використовуються описані вище операції над каскадами.

Основу виведення утворює банк стратегій, який включає в себе існуючі моделі виведення: метод резолюцій, дедукцію та ін., в тому числі моделі виведення, характерні для процедур проектування.

Основними операціями, визначеними на множині стратегій, є: запуск продукцій; означивание продукцій; услов-перехід; визначення стратегії; завершення виведення.

2.1 Формат метаданих

Для спрощення розуміння та вивчення синтаксису, зменшення бар'єру входу нових розробників за основу було вирішено прийняти синтаксис опису об’єктів з мови програмування ECMAScript (JavaScript). Оскільки ECMAScript є досить популярною та простою мовою програмування, більшість програмістів та бізнес аналітиків, вже будуть ознайомлені з синтаксисом опису даних, крім того ECMAScipt є однією з найбільш сучасних та гнучких мов програмування.

В ході проектування метаданих було виділено такі види сутностей:

Домени — опис скалярних типів, що будуть використовуватись в системі. Призначені для опису базових типів даних у системі. Являються загальносистемними та незалежними від контексту використання. Прикладами доменів будуть являтись такі типи як “рядки”, “числа”, “числа” з обмеженнями за мінімумом, максимумом, точністю, “дати” та ін.

Категорії — опис комплексних типів, що будуть використовуватись в системі. Призначені для опису безпосередньо сутностей у системі в контексті їх використання та бізнес-умов. Можуть бути як простими структурами з даними (наприклад, опис транзакції включаючи її номер та час виконання) так і поєднувати між собою велику кількість інших сутностей складними зв’язками.

Дії — специфічні сценарії операцій, що виходять за рамки операцій базового створення, читання, оновлення та видалення. Дані операції зазвичай оперують над елементами категорій, виконують складні бізнес процеси, або дозволяють підвищити ефективність отримання інформації. Крім того існує можливість створювати дії для модифікації базових CRUD операції за необхідності. Кожна дія може, за необхідності, виконувати запити з правами користувача або з системними правами.

2.2.1 Опис синтаксису схем

Для опису схем було використано РБНФ нотацію з наступними розширеннями:

Посдіовність \символ використовуються для використання в місцях де синтаксис використовує елементи, які зарезервовані нотацією РБНФ ({}[],)
Додано термінальні символи ідентифікатор, число, логічне_значення, рядок синтаксис яких відповідає відповідним літералам в ECMAscript.

Домени

Для опису доменів використовуються схеми з розширенням *.domains

```
Домени ::= \{
  { Домен\, }
\}
Домен ::= ідентифікатор: \{
  type: 'number'|'string'|'boolean'|'bigint'|'object'\,
  [subtype: 'int'|'double'\,]
  [class: ідентифікатор\,]
  [min: число\,]
  [max: число\,]
  [sensitive: логічне_значення\,]
  [parse: js-функція\,]
  [check: js-функція\,]
\}
```

Таблиця 2.1 — Опис характеристик домену

Характеристика	Опис
type	Тип скалярного значення
subtype	Підтип чисел цілі та з плаваючою крапкою.
class	Класс, для значень з типом ‘object’

min	Мінімальне значення
max	Максимальне значення
sensitive	Помічає дані як “чутливі”, такі данні не будуть відображатись в логах системи
parse	Функція, яка приймає серіалізованне значення і трансофрмує його у відповідно репрезентацію в пам’яті процесу
check	Функція, що використовується для валідації можливих значень

Категорії

Для опису категорій використовуються схеми з розширенням {назва}.category

```
Категорія ::= [Декоратор]\(\{
  { Запис\, }
\}\)

Запис ::= ідентифікатор: Поле|Декорована_характеристика

Поле ::= \{
  domain|category: рядок\,
  [required: логічне_значення\,]
  [index|unique: логічне значення\,]
  [lookup: логічне_значення\,]
  [validate: js-функція\,]
\}

Декорована_характеристика ::= Декоратор({аргументи})
```

Таблиця 2.2 — Опис характеристик категорії

Характеристика	Опис
domain	Назва домену, екземпляром якої буде поле
category	Назва категорії, на екземпляр якої посилатиметься поле
required	Ознака обов’язковості заповнення поля
index	Ознака необхідності будувати індекс за цим полем, для пришвидшення пошуку
unique	Ознака унікальності значеннння серед всіх екземплярів категорії
lookup	Ознака того що це поле необхідно використовувати як індентифікатор в інтерфейсі користувача, при посиланні на цю категорію
validate	Функція, що використовується для валідації можливих значень

Дії

Для опису дій в системі використовуються файли {категорія}.{дія}.action.

```
Категорія ::= Action\(\{
  Execute: js-функція\,
  [Args: \{
    { Запис\, }
  \}\,]
  [Returns: \{
```

```
    { Запис\, }
  \}\,]
\}\)
```

Таблиця 2.3 — Опис характеристик категорії

Характеристика	Опис
Execute	Функція, яка виконує необхідні дії, сигнатура функціїї наступна: provider — провайдер доступу до даних context — контекст сесії користувача, може використовуватись для збереження данних про користувача, та стану, для виконання складних багатокрокових дій args — аргументи, які передав користувач
Args	Опис аргументів функції
Returns	Опис даних, які повертає функція

2.2.2 Декоратори

Декоратори - це особливі функції що надаються системою для позначення даних, створення сутностей особливого формату або спрощення синтаксису для часто використовуваних шаблонів. Основні назви декораторів сутностей представлені в таблиці 2.4.

Таблиця 2.4 — Декоратори сутностей

№	Назва Декоратора	Опис Декоратора
1	Enum()	використовується для створення перелічуваних типів, приймає список значень

2	Flags()	використовується для створення перелічуваних типів у форматі флагів, приймає список значень
3	Registry()	використовується для створення реєстрів
4	Dictionary()	використовується для створення словників
5	Local()	використовується для створення даних, що будуть зберігатись лише на 1 сервері, як правило це (detail записи)
6	History()	використовується для створення даних, для яких потрібно зберігати історію всіх станів
7	Many()	використовується для створення зв'язків багато-до-багатьох
8	Master()	використовується для створення detail-записів, вказується назва master-запису
9	Catalog()	використовується для вказання, що дане поле є признаком того, що категорія ділиться по каталогам

Продовження таблиці 2.4 — Декоратори сутностей

10	Subsystem()	використовується для вказання, що дане поле є признаком того, що категорія ділиться по підсистемам
11	Include()	використовується, щоб включити в категорію групу полів (описується як категорія)
12	Index()	використовується для визначення індексів
13	Unique()	використовується для визначення унікальних індексів
14	Execute()	використовується для визначення того, що після даної дії необхідно виконати іншу

15	Action()	використовується для визначення Дій
16	LogStatus.warning()	використовується для попередження адміністратора про результат виконання Дії
17	LogStatus.critical()	використовується для попередження адміністратора про підозрілу активність
18	Application()	використовується для опису додатків
19	Group()	використовується для опису групи елементів меню
20	Hierarchy()	використовується для опису сутностей, котрі ієрархічно діляться самі по собі
21	List()	використовується для опису списку сутностей
22	Log()	використовується для опису категорій котрі є логами, а не даними

Продовження таблиці 2.4 — Декоратори сутностей

23	Memory()	використовується для опису даних, які зберігаються в пам’яті
24	Validate()	в нього можна передати функцію, котра буде валідувати об’єкт
25	View()	сутність аналогічна в’юхам в БД
26	System()	використовується для опису системних таблиць
27	Table()	вказує, що джерелом даних є існуюча таблиця

2.3 Спосіб генерації користувацьких інтерфейсів

В корпоративних системах завжди присутня величезна кількість складно пов'язаних даних які необхідно правильно та зручно відображати. Завдяки

особливостям схем категорій та їх декларативності можливо автоматизувати створення базових варіантів інтерфейсів для їх перегляду, створення та зміни. Метадані вже наявні у схемах використовуються для генерації інтерфейсу на льоту безпосередньо у клієнта що дозволяє значно прискорити процес створення та тестування схем виключаючи необхідність у втручанні спеціалізованих фахівців. Крім того надається можливість створення власних інтерфейсів для показу та зміни даних використовуючи декларативный синтаксис. Що може бути використано для спеціалізованих сценаріїв використання або спрощення інтерфейсу де в цьому присутня необхідність.

2.4 Спосіб генерації шару доступу до даних

Аналогічно з прикладом користувацьких інтерфейсів завдяки декларативності опису даних присутня можливість автоматичної генерації SQL скриптів для створення, оновлення та міграції класичних реляційних баз даних. Запропонований формат доменів і категорій а також зв'язків між ними аналогічний спрощеній реляційній моделі та може бути проведений до неї автоматично. Для доменів створюються за необхідності типи у базі даних. Кожна категорія буде зазвичай відповідати одній або декільком таблицям в залежності від зв’язків які задані у схемі.

2.5 Масштабування

Для гарантування можливості масштабування використовується унікальний глобальний ідентифікатор. Ідентифікатором об’єкту є безрозмірне ціле число (на перших етапах може використовуватись 64- чи 128-бітне ціле число). Даний ідентифікатор є унікальним в межах системи (не може бути іншого об’єкту з таким ідентифікатором).

При розподілі даних між серверами виникає проблема синхронізації змін між кожним з серверів, в межах СКБД GlobalStorage обрано стратегію розв’язання цієї проблеми, яка заключається в існуванні Master-серверів кожен з яких відповідав би за свою частину даних. Для того щоб оновити будь-який

запис в БД — необхідно надіслати запис на Master-сервер цих даних, який в свою чергу сповістить всі сервери, що зберігають резервні копії цих даних. Це дозволяє в кожний конкретний момент часу гарантовано отримати актуальний стан будь-якого об’єкту, звернувшись до Master-серверу цих даних. Недоліком даного підходу є те, що він створює нову проблему — пошук Master-серверу.

Пошук Master-серверу може бути реалізований в два способи:

існування реєстру, який би зберігав відповідність між даними та їх Master-серверу та резервних серверів;

інформація про Master-сервер повинна міститись в самому ідентифікаторі.

У використання реєстру існує ряд недоліків:

вразливий вектор атаки;

необхідність додаткової мережевої взаємодії для виявлення серверу, відповідальному за об’єкт.

В зв’язку з цим було вирішено вибрати інший підхід — ідентифікатор об’єкта буде містити інформацію про Master-сервер. Реалізувати цей метод було вирішено у наступний спосіб: молодші біти ідентифікатора будуть визначати ідентифікатор серверу, що відповідає за ці дані. Для реалізації можливості динамічно змінювати інфраструктуру серверів було вирішено визначати на основі поточної карти серверів, яка буде знаходитись на кожному з них. Карта серверів — це об’єкт що містить відповідність між ідентифікатором серверів організованих в деревоподібну структуру, приклад якої зображено но рис.1 (синім виділено вінцеві вузли, сіри - “віртуальні”).

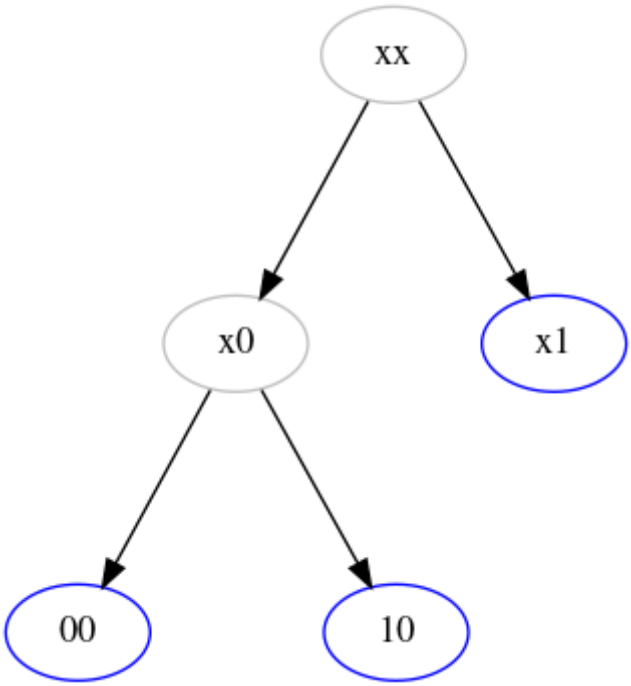

Рисунок 2.1 — Приклад інфраструктури серверів

При наведеній структурі всі об’єкти ідентифікатор яких закінчується на 1 будуть розташовані на сервері з ідентифікатором 1. Об’єкти ідентифікатор яких закінчується на 0 будуть розташовані на сервері з ідентифікатором 00 чи 10 в залежності від наступного біта в ідентифікаторі.

Якщо виникає необхідність розділити сервер, створюються два сервери ідентифікатор яких буде відповідно 0 чи 1 додані зліва до ідентифікатора існуючого сервера, приклад на рис. 2. Такий спосіб дозволяє необмежено ділити сервери.

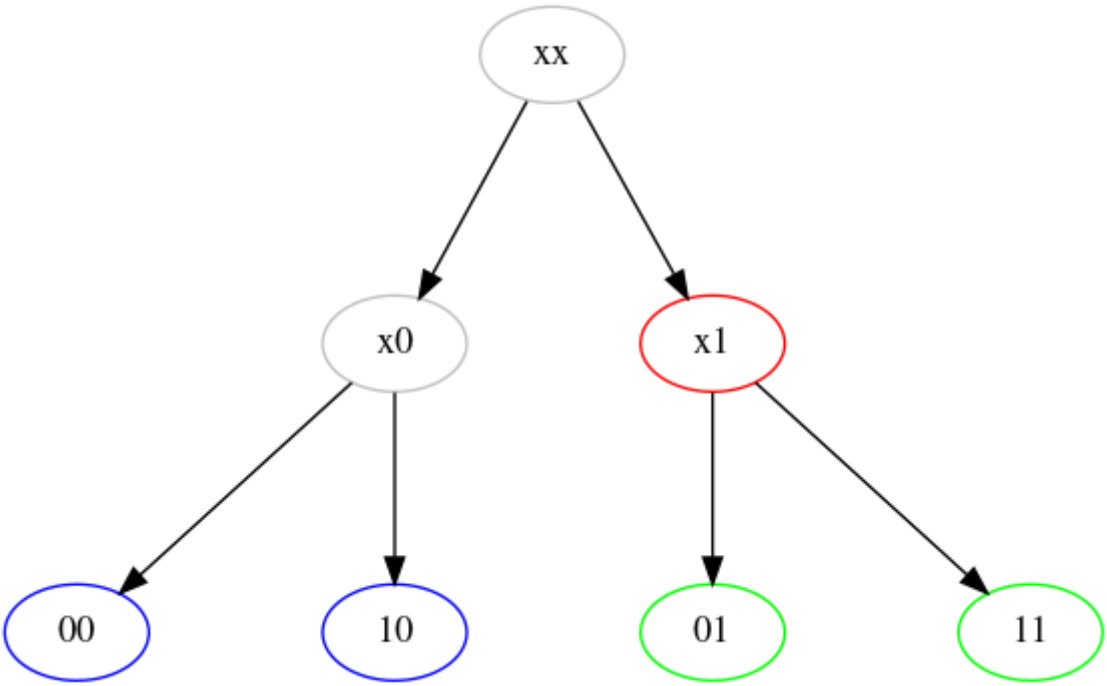

Рисунок 2.2 — Приклад ділення серверу

2.6 Управління правами доступу до об’єктів

Існує дві моделі управління правами доступу: основана на дозволах і заборонах. При використанні моделі основаною на заборонах, користувач, за замовчуванням, має доступ до всіх об’єктів системи, доступ до яких йому не заборонили. В моделі побудованій на дозволах користувач має доступ лише до тих, об’єктів, на які йому явно надали доступ. Було вирішено вирокистовувати модель основану на дозволах, так як це зменшить ймовірність випвлкового надання новому користувачу прав на об’єкти, які не повинні бути йому доступні.

Для зменшення дублікації однотипних прав, було прийнято рішення використовувати рольову модель прав, коли користувачу присвоюється перелік ролей, кожна з яких має доступ до ввірених їй даних.

Задля можливості обмежувати права користувачів над даними в одній категорії, було введено поняття каталогів і підсистем, до яких можуть належати дані. Якщо категорія підлягає каталогізації та діленню на підсистеми — для виконання операції над конкретним записом користувачу необхідно мати право виконання цієї операції у каталозі та\або підсистемі до яких належить запис. Щоб уникнути комбінаторного вибуху прав на записи, що діляться по

каталогами та підсистемам, було вирішено зберігати окремо права на підсистему, та окремо на каталоги. Замість того, щоб видавати право на виконання операцій у вказаному каталозі у вказаній підсистемі, ролі надається право виконання операції у вказаному каталозі та право виконання операції у вказаній підсистемі.

При першому запуску системи створюється суперадміністратор, який має права на доступ до всієї системи, такий користувач використовуватиметься для створення звичайних адміністраторів, кожен з яких матиме доступ лише до своєї частини системи, в якій вони відповідатимуть за надання прав та створення звичайних користувачів.

Висновки до розділу

Було розроблено мову опису метаданих, описано процес автоматизованої генерації користувацьких інтерфейсів, шарів доступу та збереження даних, описано процес управління правами користувачів та обгрунтовано можливість масштабування.

3 АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У даному розділі було описано архітектуру розробленого програмного забезпечення.

3.1 Архітектура програмного забезпечення

Перед розробкою ПЗ потрібно обрати ОС, платформу та технологічний стек, що будуть використані під час розробки.

За даними W3Cook від травня 2015 року, найбільш розповсюдженими ОС, що використовуються на серверах є ОС основані на Linux [9]. Також Linux є ОС з відкритим вихідним кодом, що дозволяє анулювати витрати на придбання ОС. Завдяки популярності та відкритому вихідному коду ядро Linux піддається аналізу багатьма спеціалістами, в тому числі спеціалістами з безпеки. Наступною перевагою є низькі вимоги до апаратного забезпечення. Тому в якості ОС було обрано Linux.

В якості платформи було вирішено вибрати Node.js. Node.js — платформа з відкритим кодом побудована на основі libuv, мультиплатформенної бібліотеки для асинхронного вводу-виводу, та V8, інтерпретатора мови програмування JavaScript. Серед основних переваг Node.js: висока продуктивність, мова програмування та велика кількість готових модулів, за офіційними даними понад 700 000 [10]. Сукупність цих переваг дозволяє швидко розробляти високопродуктивні додатки.

За основу було вирішено вибрати технологічний стек з відкритим кодом Metarhia, що включає в себе: сервер застосунків Impress, мережевий протокол JSTP, бібліотеку Metasync.

Impress — сервер застосунків на платформі Node.js, заснований на таких ідеях: монолітна архітектура, простота коду прикладного рівня, високий рівень абстракції, підтримка різних моделей роботи з клієнтами: зі станом та без, підтримка плагінів, використання асинхронного лінивого вводу\виводу та його мінімізація, підтримка декількох протоколів для клієнт-серверної та міжсерверної взаємодії, можливість конфігурації горизонтального

масштабування серверної інфраструктури, реалізація вбудованих спрощених систем логування та тестування, роутинг запитів, що базується на директоріях файлової системи, кешування та автоматичне оновлення при зміні вихідного коду для програмних реалізацій обробників запитів [11].

JSTP — RPC протокол, його реалізація від Metarhia у вигляді бібліотеки для Node.js та браузерів, та однойменний формат серіалізації даних. Протокол забезпечує двосторонню асинхронну передачу даних з підтримкою багатьох неблокуючих взаємодій, розроблений таким чином, щоб мінімізувати відмінності між викликом локальних асинхронних функцій та віддалених процедур. Протокол та відповідна його реалізація підтримує використання декількох поширених транспортних протоколів, а саме TCP, TLS, WebSocket, крім цього підтримуються вбудовані в операційну систему механізми міжпроцесової взаємодії, а саме локальні POSIX IPC сокети, що в UNIX подібних операційних системах реалізовані у вигляді сокетів домена Unix, а в операційних системах сімейства Windows – у вигляді іменованих каналів. Формат серіалізації даних представляє з себе строкове зображення в UTF-8 кодуванні літералів відповідного довільного типа (за виключенням функціональних) відповідно до специфікації стандарту ECMAScript [12].

В якості платформи для клієнту користувача було обрано браузер. На даний момент web-застосунки є найбільш ефективним способом розробки клієнтів користувача в зв’язку з тим, що браузер нині доступний на будь-якому ПК, смартфоні та інших мультимедійних пристроях комунікації, більш того, розробка web-застосунка дозволює використовувати одне програмне забезпечення для всіх платформ.

Для спрощення розробки програмного забезпечення було вирішено використовувати стек ReactRedux.

React - бібліотека JavaScript для створення інтерфейсів користувача. Особливості react [13]:

Декларативний: React робить безболісним створення інтерактивних інтерфейсів користувача. Створіть прості перегляди для кожного стану у вашій програмі, і

React ефективно оновлює та надасть лише потрібні компоненти, коли ваші дані зміняться. Декларативні подання роблять ваш код більш передбачуваним, простішим для розуміння та простішим налагодженням.

На основі компонентів: Створіть інкапсульовані компоненти, що управляють власним станом, а потім складіть їх, щоб скласти складні інтерфейси користувача. Оскільки логіка компонентів написана в JavaScript замість шаблонів, ви можете легко передавати багаті дані через додаток і не підтримувати стан DOM.

Навчіться один раз, пишіть де-небудь: ми не робимо припущень щодо решти вашої технології стеку, тому ви можете розробляти нові функції в React, не переписуючи наявний код. React також може відображатися на сервері за допомогою Node та живлення мобільних додатків за допомогою React Native.

Redux - це передбачуваний контейнер стану для додатків JavaScript. Це допомагає писати програми, які ведуть себе послідовно, запускаються в різних середовищах (клієнт, сервер та рідне середовище), і їх легко перевірити. Крім того, це забезпечує чудовий досвід для розробників, наприклад редагування живого коду в поєднанні з відладчиком у часі [14].

Архітектуру системи можно умовно роділити на серверну та клієнтську частини, а також бібліотеки що працюють в обох частинах.

Серверна частина складається з таких основних компонентів - Impress, JSTP, GlobalStorage та Metaschema. Impress виступає основою даної системи та забезпечує роботу інших компонентів, управління ресурсами, базою даних, логування інформації та відновлення у випадках несправностей.

JSTP працює всередині Impress Application та забезпечує постійне з’єднання з клієнтами та між серверами а також відповідає за створення та відновлення сесій. JSTP Client присутні як на серверній частині для взаємодії між серверами так і на клієнтській для з’єднання з сервером. JSTP Application це набір методів бізнес-логіки за допомогою яких відбувається двостороння взаємодія клієнта та сервера. Клієнт у такій моделі також має свій інтерфейс і сервер може виконувати запити до нього. За допомогою JSTP Application

створюється основна точка доступу для клієнтів, низькорівневі операції управління сервером та надається API для забезпечення основної взаємодії через GlobalStorage. GlobalStorage реалізує спеціальну модель JSTP Application за допомогою якої клієнт може виконувати відповідні виклики до серверу прозоро через JSTP з'єднання.

Metaschema є ключовим компонентом даної системи та безпосередньо дозволяє описувати дані та наявні запити до них у тому числі бізнес логіку рівня окремих застосунків. Вона надає спеціальні мови DSL (Domain specific language) що базуються на ECMAScript для опису необхідних структур системи та взаємодій і відношень між ними. Даний опис в подальшому використовується для створення скриптів створення та оновлення баз даних, зокрема PostgresSQL. Крім того Metaschema дозволяє описувати методи які клієнт зможе використовувати у своєму інтерфейсі. Кожен такий метод асоційований з набором даних які клієнт має надати, не обов'язковою формою для збирання цих даних (вона може бути створена автоматично з опису методу) та безпосередньо функцією на мові ECMAScript яка буде описувати бізнес-логіку. Metaschema повністю бере відповідальність за парсинг, валідацію та серіалізцію даних клієнта відповідно до описаних схем та перевірок. Кожен такий метод має доступ до сесії клієнта (якщо вона присутня) та GlobalStorage з відповідно можливістю виконувати довільні дії. База даних на момент виклику методів уже буде містити усі задані структури, таблиці та дані описані з допомогою Metaschema.

Клієнтська частина даної системи складається з фреймворку React який є архітектурною основою клієнта, Redux бібліотеки для керування станом та даними у додатку та набором класів та компонентів що відповідають за коректне відображення даних також зв'язок з сервером.

Для зв'язку з сервером використовується клієнтська частина GlobalStorage, а зокрема GlobalStorage RemoteClient який дозволяє виконувати запити до даних та бізнес логіки сервера прозорим чином з використанням

JSTP. Кожен запит до даних буде автоматично валідований та відфільтрований за доступністю користувачем.

При запуску клієнт виконує запит на сервер для отримання початкових схем Metaschema та відповідно до них ініціює інтерфейс. Клієнт використовує схеми Metaschema для генерації інтерфейсу вводу та відображення даних у автоматичному режимі або використовує наявні форми для конкретних інтерфейсів за їх наявності. Крім того перед відправкою даних до сервера клієнт проводить їх валідацію на відповідність схемі що дозволяє по перше пришвидшити отримання результату користувачеві а по друге зменшити навантаження на сервер. Також схеми використовуються для коректного відображення даних, в залежності від їх домену, типу або інших накладених обмежень.

В процесі роботи схеми можуть бути оновлені на сервері, у такому випадку клієнт отримує повідомлення від сервера та виконує запит на отримання нових схем. Після чого відповідно до них оновлює власний стан та інтерфейс, при цьому це проходить прозоро для користувача та його сесія не переривається.

Схему зв’язків між сутностями та структури Бази даних наведено у ДОДАТКУ 1 та ДОДАТКУ 2 Відповідно.

3.2 Аналіз безпеки даних

Для збереження паролів використовуються хешування паролів за допомогою алгоритму bcrypt. bcrypt був вибраний так як він є одним з алгоритмів, рекомендованих OWASP, для хешування паролів [15].

bcrypt — адаптивна криптографічна функція формування ключа, що використовується для безпечного зберігання паролів. Розробники: Нільс Провос і David Mazières. Функція заснована на шифрі Blowfish, вперше представлена на USENIX у 1999 році. Для захисту від атак за допомогою райдужних таблиць bcrypt використовує сіль (salt); крім того, функція є

адаптивною, час її роботи легко настроюється і її можна сповільнити, щоб ускладнити атаки перебором [16].

Так як GlobalStorage — це in-memory база даних, єдиний способ доступу до данних — це звернення до API серверу. Під час кожного до серверу проходить повна перевірка, що на даний момент користувач має право на виконання вказаних ним дій.

Кожна дія, яка була виконана в системі записуються в журнал змін, що дозволяє відновити систему з будь-якого стану, крім того це дозволяє проводи аудит безпеки, у випадку необхідності дізнатись хто отримував доступ до вказанних данних, також це дає можливість моніторингу доступу до специфічних даних.

3.5 Висновки до розділу

У даному розділі було виконане моделювання і аналіз програмного забезпечення. Аналіз безпеки даних показав, що спроектована система має достатній рівень захисту.

Обґрунтовано вибір технологій, наведено порівняння із аналогами.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У даному розділі було розглянуто програмну архітектуру і використані інструменти для програмної реалізації.

4.1 Аналіз якості програмного забезпечення

Тестування э одним з найважливіших етапів розробки ПЗ. Тестування серверу є дуже важливим, оскільки дуже велика частина логіки бізнес процесів відбувається на сервері. Також сервер – є єдиною частиною інформаційної системи, що має доступ до всіх даних, які знаходяться в системі.

Працездатність серверу безпосередньо впливає на працездатність всією ІС, оскільки, коли він не працює, ні один з клієнтів не може виконати більшість або й всі свої завдання.

Існує багато видів тестування, що використовують для програмного забезпечення.

Модульне тестування (англ. Unit testing) — це метод тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду програми. Модулем називають найменшу частину програми, яка може бути протестована [17].

Інтеграційне тестування (англ. integration testing) — це фаза тестування програмного забезпечення, під час якої окремі модулі програми комбінуються та тестуються разом, у взаємодії. Інтеграційне тестування виконується після модульного тестування [18].

Статичний аналіз коду (англ. static code analysis) — аналіз програмного забезпечення, який здійснюють (на відміну від динамічного аналізу) без реального виконання програм, що досліджуються [19].

Перед розгортанням серверу у робочому середовищі, необхідно провести всі види тестів та статичний аналіз коду. Після закінчення тестів необхідно сформувати звіт про тестування, що включає в себе:

список дефектів;

оцінку якості програмного продукту;

опис критичних дефектів;

висновок про подальшу роботу: розгортання в робочому середовищі,чи повернення продукту на доопрацювання.

4.2 Опис середовища тестування

Для тестування було використано віртуальний сервер, на який встановлено ОС Ubuntu 18.04, що має 4 ГБ оперативної пам'яті, 80 ГЬ постійного сховища та 2 віртуальних процессора. Було здійснено модульне й інтеграційне тестування та статичний аналіз коду. Під час виявлення помилок, її було детально документовано як окрему проблему у GitHub Issues. В якості тестового клієнту використовувався ПК на який встановлено ОС Arch linux та Google Chrome v78, що має 2 ГБ оперативної пам'яті, 40 ГЬ постійного сховища та 2 процесорні ядра.

4.2.1 Документація функціональних тестів

Для контролю якості ПЗ було розроблено тест-кейзи, їх опис на результат нтаблиці 4.1.

Таблиця 4.1 – Тести функціональності сервера

Ідентифікатор тесту	Застосунок	Опис	Статус тесту
TC–01	Сервер	Перевірити можливість запуску системи з коректними схемами	Пройдено
TC–02	Сервер	Перевірити неможливість завпуску системи з некоректними схемами	Пройдено
TC–03	Сервер	Перевірити можливість	Пройдено

		розширення бахових схем	

Продовження таблиці 4.1 – Тести функціональності сервера

ТС–04	Сервер	Перевірити можливість перезавантаження змінених схем під час роботи серверу	Пройдено
ТС–05	Сервер	Перевірити наявність повідомлення про можливість переходу на нову версію схем	Пройдено
ТС–06	Сервер	Перевірити можливість продовжувати работу зі старими версіями схем після оновлення	Пройдено
ТС–07	Сервер	Перевірити генерацію табличного простору для постійного збереження даних	Пройдено
ТС–08	Сервер	Перевірити оновлення табличного простору для постійного збереження данних	Пройдено
ТС–09	Сервер	Перевірити правильність генерації запитів до постійного сховища	Пройдено
ТС–10	Сервер	Перевірити створення суперадміністратора	Пройдено
ТС–11	Сервер	Перевірити можливість створення нових користувачів	Пройдено
ТС–12	Сервер	Перевірити можливість надати права користувачам	Пройдено
ТС–13	Сервер	Перевірити можливість забирати права користувачам	Пройдено
ТС–14	Сервер	Перевірити неможливість надавати права на ті частини системи, до якої у адміністратора немає доступа	Пройдено

Продовження таблиці 4.1 – Тести функціональності сервера

ТС–15	Сервер	Перевірити неможливість виконати запит при відсутності прав на нього	Пройдено
ТС–16	Сервер	Перевірити можливість виконати запит з корректними правами	Пройдено
ТС–17	Сервер	Перевірити можливість виконання запитів до публічних даних всіма авторизованними та неавторизованними користувачами	Пройдено
ТС–18	Сервер	Перевірити можливість блокування користувача	Пройдено
ТС–19	Сервер	Перевірити неможливість виконання операцій заблокованним користувачем	Пройдено
ТС–20	Сервер	Перевірити можливість авторизуватись з коректними даними	Пройдено
ТС–21	Сервер	Перевірити неможливість авторизуватись з некоректними даними	Пройдено
ТС–21	Сервер	Перевірити неможливість авторизуватись заблокованному користувачу	Пройдено
ТС–22	Сервер	Перевірити неможливість повторно авторизуватись	Пройдено
ТС–23	Сервер	Перевірити можливість оновити пароль	Пройдено
ТС–24	Сервер	Перевірити неможливість оновити пароль з неправильно вказаними даними	Пройдено
ТС–25	Сервер	Перевірити логування запитів	Пройдено

ТС–26	Сервер	Перевірити відсутність в логах чутливих даних	Пройдено

Продовження таблиці 4.1 – Тести функціональності сервера

ТС–27	Сервер	Перевірити можливість моніторингу запитів	Пройдено
ТС–28	Сервер	Перевірити можливість проведення аудиту	Пройдено
ТС–29	Сервер	Перевірити коректність шардування	Пройдено
ТС–30	Клієнт	Перевірити коректність підключення до серверу	Пройдено
ТС–31	Клієнт	Перевірити завантаження схем	Пройдено
ТС–32	Клієнт	Перевірити можливість оновти схеми за запитом користувача	Пройдено
ТС–33	Клієнт	Перевірити побудову атоматичну побудову інтерфейсу	Пройдено
ТС–34	Клієнт	Перевірити коректність побудови підлеглих зв’язків	Пройдено
ТС–35	Клієнт	Перевірити можливість авторизації	Пройдено
ТС–36	Клієнт	Перевірити коректність формування запитів в результаті виконання дій користувачем	Пройдено

Знайдені недоліки та висновки до розділу

Було проведено аналіз якості програмного забезпечення, а саме серверного та клієнтських застосувань.

Було визначено найважливіші етапи в тестуванні програмного забезпечення. Це модульне й інтеграційне тестування та статичний аналіз коду.

Було виявлено загальні критерії тестування і розроблено план тестування згідно сценаріям використання розроблюваної системи

У результаті тестування було розроблено план тестування і виявлено, що розроблене програмне забезпечення задовільняє всі вимоги, та готове до розгортання в робочому середовищі.

5 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Тестування програмного забезпечення

Для встановлення серверного застосунку необхідно доступ до сервера під управлінням дистрибутива Linux, на якому встановлено систему контролю версій git, Node.js, PostgreSQL, та компілятор C++.

Далі необхідно завантажити ПЗ на сервер.

Потім необхідно встановити certbot для генерації SSL-сертифікатів, дя цього необхідно виконати команди: «curl -O https://dl.eff.org/certbot-auto» та «chmod a+x certbot-auto».

Після цього необхідно згенерувати сертифікати: «./certbot-auto certonly».

Для того, щоб запустити сервер необхідно виконати команду: «./server.sh».

5.2 Робота з програмним забезпеченням

Після запуску сервер буде продовжувати працювати в автономному режимі. Під час роботи сервер буде надавати користувачам найбільш актуальну версію клієнтського застосунку. У разі необхідності можно підключитись до серверу та оновити схеми, вони будут автоматично перезавантажені сервером.

Висновки до розділу

У даному розділі було представлено необхідні умови для розгортання серверу.

ВИСНОВКИ

В рамкаъ даної роботи було розроблено мову метаданих, програмне забезпечення для автоматизованого створення інформаційних систем на основі розробленої мови програмування та описано механізм гарантування можливості горищонтального масштабування створенної системи.

Було досліджено існуючі методи виконання поставленої задачі, їх переваги та недоліки.Було розглянуто існуючі мови опису метаданиих.

Було проведено тестування програмного забезпечення на відповідність всім вимог розробленим до нього.

Було запропоновано декілька стратегій впровадження інноваційного продукту і детально розроблено стратегію впровадження продукту як частини програмного забезпечення “під ключ” для існуючої компанії.

СПИСОК ДЖЕРЕЛ

1) ISO/IEC 2382 [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: https://www.iso.org/obp/ui/#iso:std:iso-iec:2382:ed-1:v1:en.
2) Когаловский М. Р. и др. Глоссарий по информационному обществу / Под общ. ред. Ю. Е. Хохлова. — М.: Институт развития информационного общества, 2009. — 160 с.
3) Павлов А.А., Гриша С.Н., Баклан И.В. Об одном подходе к описанию информационной среды ИАСУ // Информационный сборник / ЦНИИТЄИприборостроения, 1986. - ТС-12. - Вып. 4. - С.10-11.
4) XML [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3.org/XML/.
5) XML Schema [Електронний ресурс] – Режим доступу до ресурсу: https://whatis.techtarget.com/definition/XSD-XML-Schema-Definition.
6) Підприємство Парус [Електронний ресурс] – Режим доступу до ресурсу: http://parus.ua/ua/140/.
7) Баклан И.В.. Гриша С.Н. Особенности реализации языка представления знаний в САПР интегрированных АСУ // Проблемно=ориентированнык диалоговые системы: Труды II республиканской конференции. Кн. I. Тбилиси: Мецциереба, 1987. с. 87-90.
8) Гриша С.Н., Баклан И.В. Язык представления знаний КАСКАД// Актуальные проблемы развития перспективных информаци онных технологий. М.: ВИМИ, 1987. с. 87-90.
9) Usage share of operating systems [Електронний ресурс]: (Стаття) /Wikipedia – Електрон. дан. (1 файл). – 2018 – Режим доступу: https://en.wikipedia.org/wiki/Usage_share_of_operating_systems#Public_servers_on_the_Internet. – Назва з екрана.
10) NPM [Електронний ресурс]: (Стаття) / NPM – Електрон. дан. (1 файл). – 2018 – Режим доступу: https://www.npmjs.com/. – Назва з екрана.

11) Impress [Електронний ресурс]: (Стаття) / Metarhia – Електрон. дан.(1 файл). – 2018 – Режим доступу: https://github.com/metarhia/impress. – Назва з екрана.
12) JSTP [Електронний ресурс]: (Стаття) / Metarhia – Електрон. дан. (1 файл). – 2018 – Режим доступу: https://github.com/metarhia/jstp/. – Назва з екрана.
13) React [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/facebook/react.
14) Redux [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/reduxjs/redux.
15) Password Storage Cheat Sheet [Електронний ресурс]: (Стаття) /OWASP – Електрон. дан. (1 файл). – 2018 – Режим доступу: https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet#Use_a_cryptographically_strong_credential-specific_salt. – Назва з екрана.
16) Bcrypt [Електронний ресурс]: (Стаття) / Wikipedia – Електрон. Дан. (1 файл). – 2018 – Режим доступу: https://uk.wikipedia.org/wiki/Bcrypt. – Назва з екрана.
17) Модульне тестування [Електронний ресурс]: (Стаття) / Wikipedia – Електрон. дан. (1 файл). – 2018 – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C%D0%BD%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F. – Назва з екрана
18) Інтеграційне тестування [Електронний ресурс]: (Стаття) / Wikipedia – Електрон. дан. (1 файл). – 2018 – Режим доступу: https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F. – Назва з екрана.
19) Статичний аналіз коду [Електронний ресурс]: (Стаття) / Wikipedia – Електрон. дан. (1 файл). – 2018 – Режим доступу:

https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D1%96%D0%B7_%D0%BA%D0%BE%D0%B4%D1%83. – Назва з екрана.